A New Adaptive Accrual Failure Detector for Dependable Distributed Systems

Benjamin Satzger, Andreas Pietzowski, Wolfgang Trumler, Theo Ungerer
Institute of Computer Science
University of Augsburg
D-86135 Augsburg, Germany
{satzger, pietzowski, trumler, ungerer}@informatik.uni-augsburg.de

ABSTRACT

The detection of failures in distributed environments is a crucial part for developing dependable, robust, and self-healing systems. The contribution of this paper is a new failure detection algorithm that can be described as an adaptive accrual algorithm coupled with features to increase flexiblity and decrease computation costs. Furthermore our evaluation results show a very good detection quality in the case of message losses.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network monitoring*; C.2.4 [**Computer- Communication Networks**]: Distributed SystemsDistributed applications; C.4 [**Performance of Systems**]: [reliability, availability, serviceability]

General Terms

Reliability

Keywords

accrual, algorithm, asynchronous systems, dependable systems, distributed systems, probability distribution, failure detection, fault-tolerance, heartbeat, histogram, self-healing

1. INTRODUCTION

Failure detectors are an important part of fault-tolerant distributed systems. They provide information on failures of components of these systems. Typically distributed asynchronous systems consisting of a finite set of processes are considered whereas each process has access to a local failure detector, see for example [3]. Failure detectors return a list of processes they are suspecting to have crashed.

This paper proposes a new failure detection algorithm that specifies an adaptive accrual failure detector. Accrual failure detectors decouple monitoring and interpretation. That makes them applicable to a wider area of scenarios and more adequate to build generic failure detection services.

The quality of our failure detection service has been evaluated and compared against other algorithms - the very promising results can be found in the evaluation section.

The paper is organised in five sections. Section 2 gives an overview of the state of the art of failure detectors and related work. Section 3 presents the failure detection algorithm. Then, section 4 describes the measurements of an implementation of the algorithm. Finally, section 5 concludes the paper and gives an overview of future work.

SAC '07, March 11-15, 2007 Seoul, Korea

2. STATE OF THE ART AND RELATED WORK

Completeness and accuracy. Several impossibility studies [2, 10, 6] show that perfect failure detectors cannot exist in asynchronous distributed systems. The major reason is the impossibility to distinct with certainty whether a process has failed or the communication network is just slow.

Chandra et al. [3] introduced the idea of failure detectors as an unreliable distributed oracle at which it is possible that (1) a process has failed but is not suspected as well as (2) a process is suspected but has not failed. Moreover a failure detector can change its mind for example stopping to suspect a process it previously suspected. In consequence the authors of [3] characterise failure detectors by specifying their properties regarding *completeness* and *accuracy*. Completeness refers to failure detectors eventually suspecting crashed processes, while accuracy restricts the mistakes that a failure detector can make.

Monitoring strategies. There exist two main monitoring approaches for failure detectors: *push* and *pull*. Assuming process p has a failure detector monitoring q. Using a push failure detector q has to send heartbeat messages to p. This information is used by p to draw conclusions about q's status. A simple failure detection algorithm using the push approach works as follows: q sends heartbeat messages at regular time intervals Δ_i to p. When p receives a heartbeat messages it trusts q for a certain period of time Δ_{to}. If this period elapses without receiving a newer heartbeat p starts to suspect q (see figure 1). An algorithm that uses a push failure detector can be found in [4].

In systems with a pull failure detection (e.g. [8]) the monitored node adopts a passive role. p monitors q by sending "are you still alive"-messages every Δ_i. If p doesn't receive an answer from q within a certain period of time Δ_{to}, p is suspecting q (see figure 2).

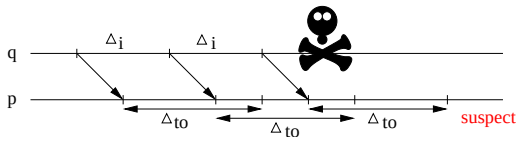

Figure 1: push failure detection

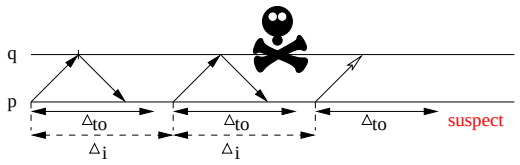

Figure 2: pull failure detection

Failure detectors using the push paradigm have some benefits compared to pull failure detectors. They need only half the messages for an equivalent failure detection quality. Furthermore it is rather hard to determine the timeout Δ_{to} as you have to take two messages into account which are both sent over the network and subject to network delays.

Adaptive failure detection. Adaptive failure detectors [5, 4, 7] are able to adjust to changing network conditions. The behavior of a network can be significantly different during high traffic times as during low traffic times regarding the probability of message loss, the expected delay for message arrivals, and the variance of this delay. Thus adaptive failure detectors are highly desirable.

Chen et al. [4] propose a well-known adaptive failure detection approach based on a probabilistic analysis of network traffic. The protocol uses sampled arrival times to compute an estimation of the arrival time of the next heartbeat. The timeout is set according to this estimation plus a constant safety margin, and is recomputed after each arrival of a new heartbeat.

Bertier et al. [1] combine Chen's estimation with another estimation developed by Jacobson [9] for a different context. Their approach is similar to Chen's - however they don't use a constant safety margin but compute it with Jacobson's algorithm.

Accrual failure detection. The principle of an accrual failure detector, introduced by Hayashibara et al. [7], is not to output whether a process is suspected to have crashed or not. Rather they give a suspicion information on a continuous scale whereas higher values indicate a higher probability that the monitored process has failed.

Hayashibara et al. propose a so-called φ failure detector that is based on an estimation of inter-arrival times assuming that inter-arrivals follow a normal distribution. They also motivate the benefits of accrual failure detectors over conventional boolean failure detectors. As principal merit they indicate that accural failure detectors favour a nearly complete decoupling between application requirements and the monitoring environment.

Lazy failure detection. Lazy failure detection protocols [5] use application messages to monitor other processes whenever this is possible.

Our algorithm can be classified as an adaptive accrual failure detector. It uses a new approach to compute suspicion information based on a histogram density estimation.

3. FAILURE DETECTION ALGORITHM

3.1 System model

We consider an asynchronous distributed system consisting of a set of n processes $\Pi = \{p_1, p_2, \ldots, p_n\}$. Each pair of processes is connected by a communication channel that can be used to send and receive messages. Processes can fail only by crashing permanently. No synchronised clocks are assumed. We also consider message loss.

3.2 Basic idea

We explain the basic concepts with an example:

Let us assume two processes, p and q, p is monitoring q, and q sends heartbeat messages to p every $\Delta_i = 1$ second. Process p manages a list S with information about the inter-arrival times of the last 1000 heartbeats it received. Let us assume $S = [1.083s, 0.968s, 1.062s, 0.993s, 0.942s, 2.037s, 0.872s, \ldots]$ at a certain point during runtime. Furthermore p stores the time of the last received heartbeat called *freshness point* f. Based on the sampled inter-arrival times and f our algorithm estimates the probability that no further heartbeat messages arrive, i.e. q has failed.

Figure 3 shows the values of S as a histogram. The shape

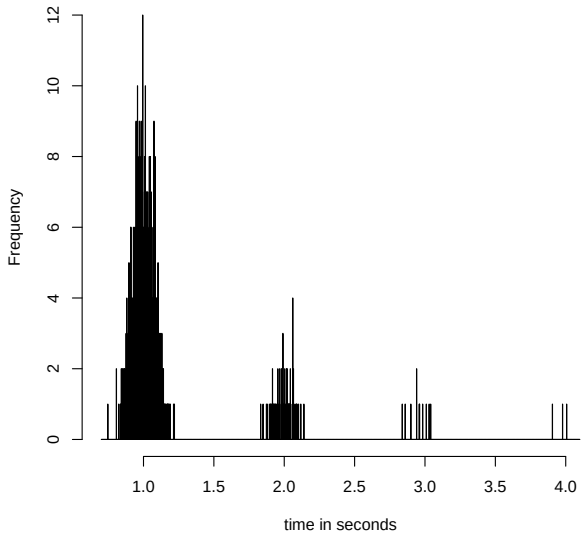

Figure 3: Histogram of the sampled inter-arrival times in S

of the histogram depends on Δ_i and the communication channel connecting q and p. In our example Δ_i is one second. The communication channel has a message loss rate of 10% and a certain fluctuating message sending delay. The peak at two seconds arises from one lost heartbeat message, the peak at three seconds arises from two consecutive lost heartbeat messages, and so forth.

This histogram can be seen as an approximation of the *probability density function* of the distribution of the inter-arrival times. Based on this histogram you can easily com-

pute the cumulative frequencies of the values in S. The cumulative frequencies in turn can be seen as an approximation of the corresponding *cumulative distribution function*. A cumulative distribution function (CDF) completely describes the probability distribution of a real-valued random variable, in our case the inter-arrival times of the heartbeat messages. The CDF $F(t_\Delta) = P(X \leq t_\Delta)$ represents the probability that an inter-arrival time takes on a value less than or equal to t_Δ. The values of the CDF are also a reasonable indicator for the crash of q: Assume p is waiting since time t for the next heartbeat from q. $F(t_\Delta) = x$ means "p is waiting for the next heartbeat message since t_Δ seconds and the probability that no further heartbeat message arrives is x". The longer p is waiting for the next heartbeat the higher are the values of F.

Figure 4 shows the cumulative frequencies of the values in S. We use these cumulative frequencies as an estimation of the real CDF of the inter-arrival times and furthermore to compute a suspicion value for the failure of q.

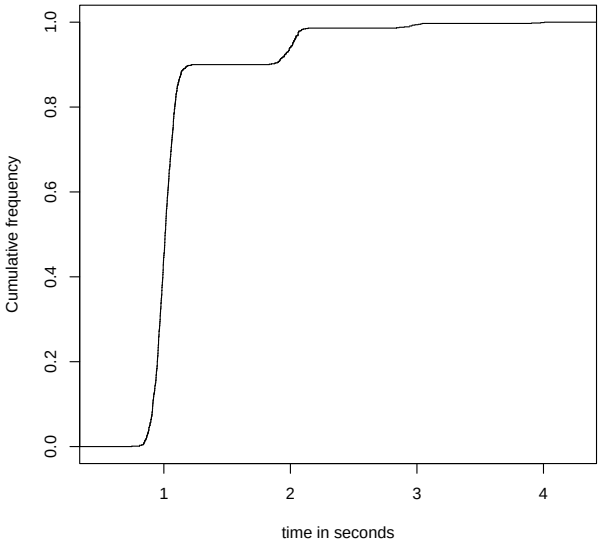

Figure 4: Cumulative frequencies of the sampled inter-arrival times in S. An approximation of the CDF F by $P_{fail,S}$.

Finally we specify the mathematical function $P_{fail,S}$ our failure detector uses to compute a suspicion value for q. It represents an estimation of the CDF of the values in S. The function simply computes the percentage of elements in S that are smaller than or equal to t_Δ. It is easy to see that this function generates a graph like in figure 4 given the list S of our example.

$$P_{fail,S}(t_\Delta) = \frac{|S^{t_\Delta}|}{|S|} \quad (1)$$

where

- $|S|$ is the number of elements in S,
- t_Δ is the time that has elapsed since the last freshness point
- $S^{t_\Delta} = \{x \in S \mid x \leq t_\Delta\}$
- $|S^{t_\Delta}|$ is the number of elements in S^{t_Δ}

3.3 The algorithm

Figure 5 shows the failure detection algorithm in detail. Again we are considering two processes p and q where q is monitored by p and its only task is to send heartbeat messages to p every Δ_i seconds. p's failure detector has the variables f, S, η and α. f is the freshness point - in our case always the time when p received the last heartbeat message. S is the list of the heartbeat message inter-arrival times. η is the maximal size of S - the so called sample window size. Whenever p receives a heartbeat the failure detector appends $t_\Delta = t - f$ (t is the current time) to S and sets $f = t$ afterwards. p removes the head of S if S grows to a size of $\eta + 1$ in consequence of appending the latest inter-arrival time.

If p wants to know the current suspicion value of q the failure detector counts the number of elements in $S^{(t_\Delta \cdot \alpha)}$ ($S^{(t_\Delta \cdot \alpha)} = \{x \in S \mid x \leq (t_\Delta \cdot \alpha)\}$) and returns its normalized value $\frac{|S^{(t_\Delta \cdot \alpha)}|}{|S|}$, where $|S|$ is the current size of S.

The scaling factor α hasn't been mentioned in the explanation of the basic idea of the algorithm. It is used to prevent the failure detector to overestimate the probability of failures particularly in the case of increasing network latency times. The computation of the suspicion value is based on the estimation of the CDF of the inter-arrival times using their cumulative frequencies.

Process q:
send heartbeat meassage to p every Δ_i

Process p:

$f = -1$ //freshness point
$S = nil$ //S is initialised as an empty list
η //max size of S (e.g. 1000)
α //a scaling factor (e.g. 1.1)

upon receive heartbeat message m_j at time t

```
if f == -1 then f = t
else
   t_Δ = t − f
   f = t
   append t_Δ to S
   if size of S > η then remove head of S endif
endif
```

on call of get_failure_probability of q at time t

$t_\Delta = t - f$
$|S^{(t_\Delta \cdot \alpha)}|$ = number elements in S that are lower or equal $(t_\Delta \cdot \alpha)$
$|S|$ = number of elements in S

return $\frac{|S^{(t_\Delta \cdot \alpha)}|}{|S|}$

Figure 5: Failure detection algorithm

4. EVALUATION

This section presents results of performance measurements of our failure detection algorithm.

4.1 Experiment setup

There exists an infinite set of environments regarding the computing devices and their interconnection in which we could test our failure detector. As we didn't want to pick one test environment we took the decision to generate the data for the evaluation. This has the benefit that the experiments are reproducible and independent of any special properties of e.g. a certain type of communication medium.

The data needed for evaluation consists of the arrival times of the heartbeat messages. We generated the arrival times as follows:

$$t_j = \begin{cases} j \cdot \Delta_i + \delta & , \quad probability : 1 - \theta \\ message\ loss & , \quad probability : \theta \end{cases} \tag{2}$$

The arrival time of the j-th heartbeat is composed of its sending time $j \cdot \Delta_i$ and the send duration δ. δ is a random variable with an associated probability distribution. θ is the probability of a message loss.

With this method we conducted two experiments. Per experiment we generated one million heartbeat messages.

For both experiments we have set the variables to the following values:

- heartbeat interval $\Delta_i = 10$ seconds
- message send duration δ is sampled according to a normal distribution with mean 0 seconds and standard deviation 0.5, $\delta = N(0, 0.5)$

The only difference between the two experiments is the message loss probability. For the first experiment we assumed no message loss ($\theta = 0\%$), and for the second experiment a message loss probability of 1%.

This choice of the values for the heartbeat generation means the following: q sends a heartbeat message to p every 10 seconds. In the first experiment p receives all the messages q has sent and the inter-arrival times are normal distributed around 10 seconds. In the second experiment heartbeat messages get lost with a probability of 1%. Thus it is not very unlikely that p has to wait about 20 seconds to receive the next heartbeat.

Within these settings we compare our failure detection algorithm with the well known failure detectors of Chen et al. [4] and Bertier [1] and the accrual φ failure detector of Hayashibara et al. [7]. We measure the algorithms' performance according to two metrics:

mistakes N_M: This measures the numbers of wrong suspicions.

detection time T_D: This is the time that elapses since the crash of q until p starts to suspect q permanently.

More information about quality metrics for failure detectors can be found in [4].

Being able to compare the accrual and non-accrual failure detectors we have to transform the accrual failure detectors into conventional failure detectors. Therefore you just have to choose a threshold T. If the level of suspicion for q is lower than this threshold then q is not suspected to have failed. If the level of suspicion crosses T then q is assumed to have crashed.

The next barrier to compare the algorithms are their different tuning parameters. These influence the time when a failure detector starts/ends to suspect a process. For the accrual failure detectors the tuning parameter is the threshold T. For Chen's failure detector the tuning parameter is the safety margin α. This is a constant period of time that is added to the estimated heartbeat arrival time. The failure detector of Bertier has no tuning parameters. Being able to compare the different failure detection algorithms we measure the behaviour of each of the failure detectors using several values of their respective tuning parameters.

To compute the detection time of the failure detectors we assume that a crash would occur exactly after successfully sending a heartbeat message. Then we measure the time it takes until the failure detector reports a suspicion. This corresponds to the worst case situation. This method to compute the worst-case detection time has also been used in [7]. To evaluate the number of mistakes we run exactly the same experiment but assume that no single crash occurs.

Finally we set the window size for all algorithms and experiments to 1000 samples. This means that the computations of the failure detectors rely only on the last 1000 heartbeat message samples. Furthermore we start our measurements not until 1000 heartbeats have been received to grant a warmup phase.

4.2 Results

The results of the performance measurements of the first experiment are depicted in figure 6. The figure shows the detection time on the horizontal axis and the mistakes on the vertical axis. Values near to the lower left corner represent a short detection time with few mistakes. As Bertier's failure detector has no tuning parameters it is only one single point in the chart.

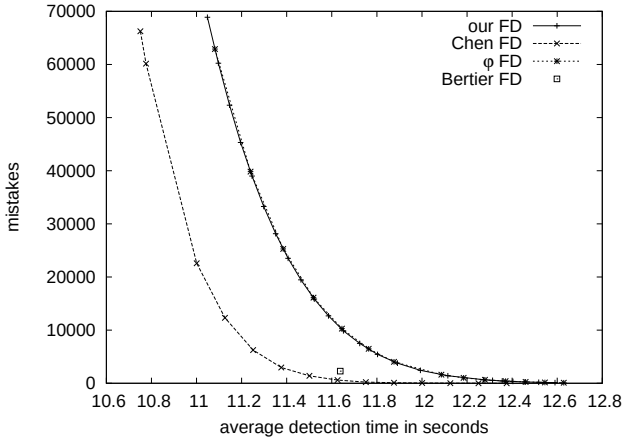

Figure 6: Results of the first experiment (no message loss). Most desirable values are toward the lower left corner. "FD" is an abbreviation for "failure detector".

Figure 7 shows the results of the second experiment.

As you can see the failure detectors of Chen and Bertier outperform our and Hayashibara's failure detector in the first experiment. Their average detection time is about 0.5 seconds shorter in this case. Our failure detector behaves nearly exactly the same than the φ failure detector. This is remarkable because the φ failure detection algorithm assumes normal distributed inter-arrival times and the inter-

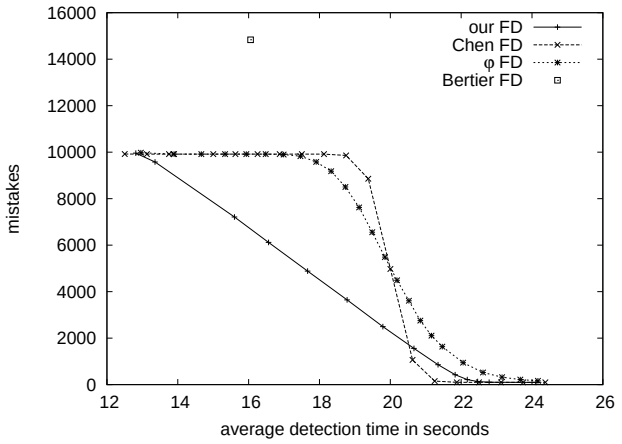

Figure 7: Results of the second experiment (1% message loss).

arrival times in the first experiment are sampled according to a normal distribution. This should be the optimal setting for Hayashibara's φ failure detector.

The results of the second experiment paint a different picture. Bertier's failure detector seems to have some difficulties coping with the message losses. This result aligns with the results of Hayashibara [7]. However you have to note that Bertier's failure detector was rather designed for LANs where messages are rarely lost.

Our failure detector outperformes the φ failure detector and performes rather well compared to Chen's algorithm. Sometimes in the second experiment our failure detector needs about $5s$ less detection time for an equivalent mistake rate compared to the other failure detectors.

Moreover our failure detector is more flexible than Chen's, Bertier's, and other non-accrual failure detectors.

The φ failure detector of Hayashibara et al. [7] is also very flexible but the computation of a suspicion value includes the following steps:

1. Determining the mean μ of the sampled values
2. Determining the variance σ^2 of the sampled values
3. Computation of

$$P_{later}(t) = \frac{1}{\sigma\sqrt{2\pi}} \int_{t}^{+\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx$$

4. Computation of $\varphi(t_{now}) = -log_{10}(P_{later}(t_{now} - T_{last}))$

This is computationally much more costly than generating a suspicion value with our algorithm, which is particularly important for systems with limited computational power. With the usage of an adequate data structure for the sampled inter-arrival times S our algorithm computes a suspicion value in $\mathcal{O}(log\ |S|)$. Furthermore the φ failure detector is restricted to environments with roughly normal distributed heartbeat inter-arrival times. Our failure detector doesn't have this limitation.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a new failure detection algorithm. It is an adaptive accrual failure detector and is thus applicable to a wide area of scenarios and adequate to build generic failure detection services. Furthermore it makes very low demands on the computational power of the hardware and is mathematically founded.

We also made performance measurements and compared our failure detector to the well-known failure detectors of Chen et al. [4] and Bertier et al. [1] and the accrual failure detector of Hayashibara et al. [7]. The measurements show that the performance of our failure detector in our experiment with message loss is excellent.

As it goes beyond the scope of this paper some important aspects couldn't be included, e.g. lazy monitoring, optimisations of the failure detection algorithm, more detailed evaluations and the motivation why our failure detector belongs to the class of Eventually Perfect Failure Detectors $\Diamond P$ [3] if the underlying system satisfies the partial synchrony assumption. These issues are work in progress and will be addressed in subsequent publications.

6. REFERENCES

[1] M. Bertier, O. Marin, and P. Sens. Implementation and performance evaluation of an adaptable failure detector. In *DSN '02*, pages 354–363, Washington, DC, USA, 2002. IEEE Computer Society.

[2] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, 1996.

[3] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

[4] W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2000)*, New York, 2000. IEEE Computer Society Press.

[5] C. Fetzer, M. Raynal, and F. Tronel. An adaptive failure detection protocol. In *PRDC '01: Proceedings of the 2001 Pacific Rim International Symposium on Dependable Computing*, page 146, Washington, DC, USA, 2001. IEEE Computer Society.

[6] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[7] N. Hayashibara, X. Défago, R. Yared, and T. Katayama. The f accrual failure detector. In *SRDS*, pages 66–78. IEEE Computer Society, 2004.

[8] M. Horstmann and M. Kirtland. Dcom architecture. Technical report, http://msdn.microsoft.com/library/backgrnd/html/msdn_dcomarch.htm, July 1997.

[9] V. Jacobson. Congestion avoidance and control. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 314–329, New York, NY, USA, 1988. ACM Press.

[10] N. Lynch. A hundred impossibility proofs for distributed computing. In *PODC '89: Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 1–28, New York, NY, USA, 1989. ACM Press.